Hash

一维

```
ll Prime_Pool[]={1998585857ll,2333333333ll};
ll Seed_Pool[]={911,146527,19260817,91815541};
ll Mod_Pool[]={29123,290182597,998244353
,1000000009,429496729ll,2333333337ll};
struct Hass{
        ll bas[N],mod;
        ll sum[N];
        Hass(ll _bas,ll _mod){
                bas[0]=1;
                mod=_mod;
                for(int i=1;i<N;i++){
                        bas[i]=(bas[i-1]*_bas)%mod;
                }
        }
        void init(char c[],int len){
                for(int i=1;i<=len;i++){
                        sum[i]=(sum[i-1]*bas[1]+c[i])%mod;
                }
        }
        ll getHash(int l,int r){
                ll ans=sum[r]-sum[l-1]*bas[r-l+1]%mod;
                if(ans<0)ans+=mod;
                return ans;
        }
}hs(Seed_Pool[0],Mod_Pool[2]);
```

二维

```
ll Prime_Pool[]={1998585857ll,2333333333ll};
ll Seed_Pool[]={911,146527,19260817,91815541};
ll Mod_Pool[]={29123,290182597,998244353,1000000009,429496729ll,2333333337ll};
char a[N][N];
struct Hass{
        __int128 sum[N][N];
        ll bas1[N],bas2[N];
        ll mod;
        Hass(ll a,ll b,ll c){
                bas1[0]=1;
                bas2[0]=1;
                mod=c;
                for(int i=1;i<N;i++){
                        bas1[i]=(bas1[i-1]*a)%mod;
                        bas2[i]=(bas2[i-1]*b)%mod;
                }
        }
        void insert(int n,int m){
                for(int i=1;i<=n;i++){
                        for(int j=1;j<=m;j++){
                                sum[i][j]=(sum[i-1][j]*bas1[1]+sum[i][j-1]*bas2[1]-sum[i-1][j-1]*ba
                                sum[i][j]=(sum[i][j]+(a[i][j]-'a'))%mod;
                        }
                }
        }
        ll query(int x1,int y1,int x2,int y2){
                x1--,y1--;
                ll ans=(sum[x2][y2]-sum[x1][y2]*bas1[x2-x1]-sum[x2][y1]*bas2[y2-y1]+sum[x1][y1]*bas
                ans=(ans+mod)%mod;
                return ans;
        }
}hs(Seed_Pool[0],Seed_Pool[1],Mod_Pool[2]),hid(Seed_Pool[0],Seed_Pool[1],Mod_Pool[2]);
```

KMP

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn=1e6+9;
const int mod=1e9+7;
int nex[maxn];
char str[maxn],ptr[maxn];
//预处理nex数组
//nex[i]为该位置的boarder
void get_next(char str[],int nex[],int len){
        nex[1]=0;
        int k=0;
        for(int i=2;i<=len;i++){
                while(k&&str[i]!=str[k+1])k=nex[k];
                if(str[i]==str[k+1])k++;
                nex[i]=k;
        }
}
int n,m;
ll num[maxn];
int main(){
        cin>>n;
    while(~scanf("%s",str+1)){
        int len=strlen(str+1);
                get_next(str,nex,len);
        ll ans=1;
        for(int i=1;i<=len;i++){
                num[i]=num[nex[i]]+1;
                }

                //快速跳nex，到 最靠近长度为一半，但<=一半的地方
        for(int i=1;i<=len;i++){
                int now=i;
                while(now*2>i){
                        if(nex[now]>now/2){
                                int period=now-nex[now];
                                now=now%period+(i/period-((now%period)?1:0))/2*period;
                                }else now=nex[now];
                        }
                        ans=(ans*(num[now]+1))%mod;
                }
                printf("%lld\n",ans);
        }
```

```
    return 0;
}
```

exkmp

给定两个字符串a,b,你要求出两个数组：

- b 的 z 函数数组 z，即 b 与 b 的每一个后缀的 LCP 长度。
- b 与 a 的每一个后缀的 LCP 长度数组 p。

对于一个长度为 n 的数组 a，设其权值为$xor_{i=1}^{n} i * (a_i + 1)$

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn=2e7+9;
ll read(){
        ll ans=0,f=1;
        char c=getchar();
        while(c<'0'||c>'9'){
                if(c=='-')f=-1;
                c=getchar();
        }
        while(c>='0'&&c<='9'){
                ans=(ans<<1)+(ans<<3)+c-'0';
                c=getchar();
        }
        return ans*f;
}
int n;
char str[maxn],ptr[maxn];
int z[maxn],p[maxn];
ll ans1,ans2;
void get_z(char s[],int z[],int len){
        int r=0,id;
        z[1]=len;
        for(int i=2;i<=len;i++){
                if(i<=r){
                        z[i]=min(z[i-id+1],r-i+1);
                }
                while(i+z[i]<=len&&s[z[i]+1]==s[i+z[i]])z[i]++;
                if(i+z[i]-1>r){
                        r=i+z[i]-1;
                        id=i;
                }
        }
}
void exkmp(char str[],int slen,int p[],char ptr[],int plen,int z[]){
        int r=0,id;
        for(int i=1;i<=slen;i++){
                if(i<=r)p[i]=min(z[i-id+1],r-i+1);
                while(i+p[i]<=slen&&str[i+p[i]]==ptr[p[i]+1])p[i]++;
                if(i+p[i]-1>r){
                        id=i;
                        r=i+p[i]-1;
                }
        }
}
int main(){
        scanf("%s",str+1);
```

```
        scanf("%s",ptr+1);
        int slen=strlen(str+1),plen=strlen(ptr+1);
        get_z(ptr,z,plen);
        exkmp(str,slen,p,ptr,plen,z);
        for(int i=1;i<=plen;i++){
                ans1^=1ll*i*(z[i]+1);
        }
        for(int i=1;i<=slen;i++){
                ans2^=1ll*i*(p[i]+1);
        }

        cout<<ans1<<endl<<ans2<<endl;
        return 0;
}
```

```
输入
aaaabaa
aaaaa

输出
6
21
```

KMP的失配树

定义 **Border**(s) 为对于 i∈[1,|s|)，满足$pre^i = suf^i$ 的字符串 pre^i的集合。

Border(s) 中的每个元素都称之为字符串 s 的 border

对于一个String *s*,有m组询问，每组询问给定整数*p*，*q*，求*s*的*p*前缀和*q*前缀的**最长公共**border的长度。

```
char str[maxn];
int nex[maxn];
void get_nex(char c[]){
        int len=strlen(c+1);
        int k=0;
        nex[1]=0;
        for(int i=2;i<=len;i++){
                while(k&&c[k+1]!=c[i])k=nex[k];
                if(c[k+1]==c[i])k++;
                nex[i]=k;
        }
}
struct line{
        int to,nex;
}l[maxn];
int fir[maxn],cntline;
void addline(int fr,int to){
        cntline++;
        l[cntline].to=to;
        l[cntline].nex=fir[fr];
        fir[fr]=cntline;
}
int siz[maxn],fa[maxn],top[maxn],wson[maxn],dep[maxn],dft[maxn];
int dftcnt;
void dfs1(int now,int f,int dp){
        fa[now]=f;
        siz[now]=1;
        dep[now]=dp;
        wson[now]=maxn-2;
        for(int i=fir[now];i;i=l[i].nex){
                int to=l[i].to;
                dfs1(to,now,dp+1);
                siz[now]+=siz[to];
                if(siz[wson[now]]<siz[to]){
                        wson[now]=to;
                }
        }
}
void dfs2(int now,int f){
        top[now]=f;
        dft[now]=++dftcnt;
        if(wson[now]==maxn-2)return;
        else dfs2(wson[now],f);
        for(int i=fir[now];i;i=l[i].nex){
                int to=l[i].to;
                if(to==wson[now])continue;
                dfs2(to,to);
        }
```

```
}
int LCA(int a,int b){
        if(a==0||b==0)return 0;
        while(top[a]!=top[b]){
                if(dep[top[a]]<dep[top[b]])swap(a,b);
                a=fa[top[a]];
        }
        return dep[a]>dep[b]?b:a;
}

int main(){
        scanf("%s",str+1);
        get_nex(str);
        int len=strlen(str+1);
        for(int i=1;i<=len;i++){
                addline(nex[i],i);
        }
        dfs1(0,0,1);
        dfs2(0,0);
        int q;
        scanf("%d",&q);
        while(q--){
                int a,b;
                scanf("%d %d",&a,&b);
                a=nex[a];b=nex[b];
                int ans=LCA(a,b);
                printf("%d\n",ans);
        }
        return 0;
}
```

输入

```
aaaabbabbaa
5
2 4
7 10
3 4
1 2
4 11
```

输出

```
1
1
2
0
2
```

输入

```
zzaaccaazzccaacczz
3
2 18
10 18
3 5
```

输出

```
1
2
0
```

Manacher

```
int n;
char c[maxn];
char tmp[maxn];
int p[maxn];
void manache(char c[],int p[]){
        int len=strlen(c+1);
        int cnt=-1;
        tmp[++cnt]='$';
        tmp[++cnt]='$';
        for(int i=1;i<=len;i++){
                tmp[++cnt]=c[i];
                tmp[++cnt]='$';
        }
        int r=0,id=0;
        for(int i=1;i<=cnt;i++){
                if(i<r){
                        p[i]=min(r-i,p[(id<<1)-i]);
                }else p[i]=1;
                while(tmp[i-p[i]]==tmp[i+p[i]]){
                        p[i]++;
                }
                if(i+p[i]>r){
                        r=i+p[i];
                        id=i;
                }
        }
        int ans=0;
        for(int i=1;i<=cnt;i++){
                ans=max(ans,p[i]);
        }
        cout<<ans-1<<endl;
}
int main(){
        scanf("%s",c+1);
        manache(c,p);
        return 0;
}
```

PAM

Given a string s, find the number of ways to split s to substrings such that if there are k substrings (p1, p2, p3, ..., pk) in partition, then pi = pk - i + 1 for all i (1 ≤ i ≤ k) and k is even.

```
char c[N],s[N];
struct PAM{
        int son[N][26];
        int len[N],fail[N];
        //int num[N]; num[i]:节点i代表的回文串的后缀回文串数量（包括自己）
        //int siz[N]; siz[i]:节点i代表的回文串在原串出现的次数
        //for(int i=cnt;i>1;i--)siz[fail[i]]+=siz[i];
        ll diff[N],anc[N],g[N],f[N];
        //diff:i与fail[i]长度相差值
        //anc:等差数列的头节点的fail节点
        //g：树上等差数列的转移和
        //f:字符串的dp转移

        int cnt=1;
        void init(){
                cnt=1;
                fail[0]=fail[1]=1;
                len[1]=-1;
                f[0]=1;
        }
        int getfail(int now,int i){
                while(c[i-len[now]-1]!=c[i]){
                        now=fail[now];
                }
                return now;
        }
        int insert(char c,int las,int i){
                int now=getfail(las,i);
                int sonid=c-'a';
                if(son[now][sonid]==0){
                        int q=++cnt;
                        fail[q]=son[getfail(fail[now],i)][sonid];
                        son[now][sonid]=q;
                        len[q]=len[now]+2;
                        diff[q]=len[q]-len[fail[q]];
                        anc[q]=diff[q]==diff[fail[q]]?anc[fail[q]]:fail[q];
                        //num[q]=num[fail[q]]+1;
                }
                las=son[now][sonid];
                //siz[las]++;
                return las;
        }
        void trans(int las,int i){
                for(int now=las;now>1;now=anc[now]){
                        g[now]=f[i-len[anc[now]]-diff[now]];
                        if(diff[now]==diff[fail[now]]){
                                g[now]=(g[now]+g[fail[now]])%mod;
                        }
```

```
                        if(i%2==0){
                                f[i]=(f[i]+g[now])%mod;
                        }
                }
        }
        void insert(char c[]){
                int len=strlen(c+1);
                int las=0;
                for(int i=1;i<=len;i++){
                        las=insert(c[i],las,i);
                        trans(las,i);
                }
                printf("%lld\n",f[len]);
        }
}pam;



int main(){
        scanf("%s",s+1);
        int len=strlen(s+1);
        for(int i=1;i<=len/2;i++){
                c[(i<<1)-1]=s[i];
                c[(i<<1)]=s[len+1-i];
        }
        pam.init();
        pam.insert(c);
        return 0;
}
```

输入：abcdcdab
输出: 1

GPAM

给定A, B 两个字符串，对于每个 A, B 的公共回文子串 S，求 $\sum resA[S] * resB[S]$的值，其中 $resA[S]$和 $resB[S]$表示串 S 在A, B中出现的次数。

两个字符串，求它们的公共回文子串对数

```
struct line{
        int to,nex;
}l[N<<1];
int fir[N],cntline;
void addline(int fr,int to){
        cntline++;
        l[cntline].to=to;
        l[cntline].nex=fir[fr];
        fir[fr]=cntline;
}
char c[N];
struct PAM{
        int son[N][26];
        int link[N],len[N];
        ll siz[N][2];
        int cnt;
        void init(){
                cnt=1;
                link[0]=link[1]=1;
                len[1]=-1;
        }
        int getlink(int now,int i){
                while(c[i-len[now]-1]!=c[i]){
                        now=link[now];
                }
                return now;
        }
        int insert(char c,int las,int i,int id){
                int now=getlink(las,i);
                int sonid=c-'A';
                if(son[now][sonid]==0){
                        int q=++cnt;
                        link[q]=son[getlink(link[now],i)][sonid];
                        son[now][sonid]=q;
                        len[q]=len[now]+2;
                }
                las=son[now][sonid];
                siz[las][id]++;
                return las;
        }
        void add(){
                for(int i=2;i<=cnt;i++){
                        addline(link[i],i);
                }
        }
        ll ans=0;
        void dfs(int now){
```

```
                for(int i=fir[now];i;i=l[i].nex){
                        int to=l[i].to;
                        dfs(to);
                        siz[now][0]+=siz[to][0];
                        siz[now][1]+=siz[to][1];
                }
                if(now<2)return;
                ans+=siz[now][0]*siz[now][1];
        }
        void solv(){
                add();
                dfs(0);
                printf("%lld\n",ans);
        }
}pam;



int main(){
        scanf("%s",c+1);
        int len=strlen(c+1);
        pam.init();
        int las=0;
        for(int i=1;i<=len;i++){
                las=pam.insert(c[i],las,i,0);
        }
        scanf("%s",c+1);
        len=strlen(c+1);
        las=0;
        for(int i=1;i<=len;i++){
                las=pam.insert(c[i],las,i,1);
        }
        pam.solv();
        return 0;
}
```

AC自动机

给你一个文本串**S**和 n 个模式串**$T_{1...n}$，请你分别求出每个模式串 T_i 在 S 中出现的次数

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=2e6+9;
struct Trie{
        int son[26],cnt;
}trie[maxn];
struct line{
        int to,nex;
}l[maxn];
int cnt,root;
int fir[maxn],fail[maxn],num[maxn],id[maxn];
void add(int fr,int to){
        cnt++;
        l[cnt].to=to;
        l[cnt].nex=fir[fr];
        fir[fr]=cnt;
}
void insert(char s[],int idd){
        int now=0;
        int len=strlen(s+1);
        for(int i=1;i<=len;i++){
                int t=s[i]-'a';
                if(trie[now].son[t]==0)trie[now].son[t]=++root;
                now=trie[now].son[t];
        }
        trie[now].cnt++;
        id[idd]=now;
}

void buildFail(){
        queue<int>q;
        for(int i=0;i<26;i++){
                if(trie[0].son[i]){
                        add(0,trie[0].son[i]);
                        fail[trie[0].son[i]]=0;
                        q.push(trie[0].son[i]);
                }
        }
        while(!q.empty()){
                int now=q.front();
                q.pop();
                //cout<<"now:  "<<now<<endl;
                for(int i=0;i<26;i++){
                        int Son=trie[fail[now]].son[i];
                        if(trie[now].son[i]){
                                //cout<<Son<<endl;
                                fail[trie[now].son[i]]=Son;
                                add(Son,trie[now].son[i]);
```

```
                                q.push(trie[now].son[i]);
                        }else{
                                trie[now].son[i]=Son;
                        }
                }
        }
}

void query(char s[]){
        int len=strlen(s+1);
        int now=0;
        for(int i=1;i<=len;i++){
                int t=s[i]-'a';
                now=trie[now].son[t];
                //cout<<"now:   "<<now<<endl;
                num[now]++;
        }
}

void dfs(int now){
        //cout<<now<<"    "<<num[now]<<endl;
        for(int i=fir[now];i;i=l[i].nex){
                int to=l[i].to;
                dfs(to);
                num[now]+=num[to];
        }
}
int n;
char str[maxn];
int main(){
        scanf("%d",&n);
        for(int i=1;i<=n;i++){
                scanf("%s",str+1);
                insert(str,i);
        }
        scanf("%s",str+1);
        buildFail();
        query(str);
        dfs(0);
        for(int i=1;i<=n;i++){
                printf("%d\n",num[id[i]]);
        }
        return 0;
}
```

SA后缀排序

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn=1e6+9;
char s[maxn];
int sa[maxn],rk[maxn],height[maxn],c[maxn],k1[maxn],k2[maxn];
/*
s[]:字符串
suff[i]:首字母下标为i的后缀
sa[i]:排名为i的后缀的首字母的下标
rk[i]:首字母下标为i的排名
height[i]:排名为i的后缀和排名为i-1的后缀的最长公共前缀
h[i]:下标为i的后缀和排名排在它前面的后缀的最长公共前缀
x[]:第一关键字
y[]:第二关键字
c[]:桶

sa[rk[i]]=i;
rk[sa[i]]=1;
LCP(i,j)=suff[sa[i]]和suff[sa[j]]的最长公共前缀;
height[i]=LCP(i,i-1);
h[i]=height[rk[i]];
height[i]=h[sa[i]];
*/
void get_SA(char s[]){
        int len=strlen(s+1);
        int m=200;
        for(int i=1;i<=m;i++)c[i]=0;
        for(int i=1;i<=len;i++)c[k1[i]=s[i]]++;
        for(int i=2;i<=m;i++)c[i]+=c[i-1];
        for(int i=len;i;i--)sa[c[k1[i]]--]=i;
        for(int k=1;k<=len;k<<=1){
                int num=0;
                for(int i=len-k+1;i<=len;i++)k2[++num]=i;
                for(int i=1;i<=len;i++)if(sa[i]>k)k2[++num]=sa[i]-k;
                for(int i=1;i<=m;i++)c[i]=0;
                for(int i=1;i<=len;i++)c[k1[i]]++;
                for(int i=2;i<=m;i++)c[i]+=c[i-1];
                for(int i=len;i;i--)sa[c[k1[k2[i]]]--]=k2[i],k2[i]=0;
                swap(k1,k2);k1[sa[1]]=1;num=1;
                for(int i=2;i<=len;i++)k1[sa[i]]=(k2[sa[i]]==k2[sa[i-1]]&&k2[sa[i]+k]==k2[sa[i-1]+k
                if(num==len)break;
                m=num;
        }
}
```

```
void get_height(char s[]){
        int len=strlen(s+1);
        int k=0;
        for(int i=1;i<=len;i++)rk[sa[i]]=i;
        for(int i=1;i<=len;i++){
                if(rk[i]==1)continue;
                if(k)k--;
                int j=sa[rk[i]-1];
                int r=max(j,i);
                while(r+k<=len&&s[i+k]==s[j+k])++k;
                height[rk[i]]=k;
        }
}

int main(){
        int tst;
        cin>>tst;
        while(tst--){
                scanf("%s",s+1);
                get_SA(s);
                get_height(s);
                int len=strlen(s+1);
                ll ans=len-sa[1]+1;
                for(int i=2;i<=len;i++){
                        ans+=len-sa[i]+1-height[i];
                }
                cout<<ans<<endl;
        }
        return 0;
}
```

后缀自动机（SAM）

给出几个由小写字母构成的单词，求它们最长的公共子串的长度。

```
struct SAM{
        int son[N<<1][26];
        int len[N<<1],link[N<<1];
        int cnt;
        int mi[N<<1],ma[N<<1];
        void init(){
                link[0]=-1;
                cnt=0;
                memset(mi,0x3f,sizeof(mi));
        }
        int insert(char c,int last){
                int now=last,p=++cnt;
                int sonid=c-'a';
                len[p]=len[now]+1;
                while(now!=-1&&!son[now][sonid]){
                        son[now][sonid]=p;
                        now=link[now];
                }
                if(now==-1)link[p]=0;
                else{
                        int q=son[now][sonid];
                        if(len[q]==len[now]+1)link[p]=q;
                        else{
                                int cop=++cnt;
                                memcpy(son[cop],son[q],sizeof(son[q]));
                                link[cop]=link[q];
                                len[cop]=len[now]+1;
                                while(now!=-1&&son[now][sonid]==q){
                                        son[now][sonid]=cop;
                                        now=link[now];
                                }
                                link[q]=link[p]=cop;
                        }
                }
                return p;
        }
        int topu[N<<1],tot[N<<1];
        void sort1(){
                for(int i=1;i<=cnt;i++)tot[len[i]]++;
                for(int i=1;i<=cnt;i++)tot[i]+=tot[i-1];
                for(int i=1;i<=cnt;i++)topu[tot[len[i]]--]=i;
        }

        void solv(char c[],int slen){
                int now=0,l=0;
                for(int i=1;i<=slen;i++){
                        int sonid=c[i]-'a';
                        while(now!=-1&&!son[now][sonid]){
```

```
                                now=link[now];
                                l=len[now];
                        }
                        if(now!=-1){
                                l++;
                                now=son[now][sonid];
                                l=min(l,len[now]);
                                ma[now]=max(ma[now],l);
                        }else{
                                now=0,l=0;
                        }
                }
                for(int i=cnt;i;i--){
                        int now=topu[i];
                        ma[link[now]]=max(ma[link[now]],min(len[link[now]],ma[now]));
                        mi[now]=min(mi[now],ma[now]);
                        ma[now]=0;
                }
        }
        int ans=0;
        void sakura(){
                for(int i=1;i<=cnt;i++){
                        ans=max(ans,mi[i]);
                }
                if(ans==1061109567)ans=0;
                printf("%d\n",ans);
        }
}sam;
int tst;
char c[N];
int main(){
        tst=read()-1;
        scanf("%s",c+1);
        int n=strlen(c+1);
        int las=0;
        sam.init();
        for(int i=1;i<=n;i++){
                las=sam.insert(c[i],las);
        }
        sam.sort1();
        while(tst--){
                scanf("%s",c+1);
                n=strlen(c+1);
                sam.solv(c,n);
        }
        sam.sakura();
        return 0;
}
```

输入
3
abcb
bca
acbc

输出
2

给定 N (N<=10^5)个字符集为小写英文字母的字符串，所有字符串的长度和小于 10^5，求出每个字符串「独特值」(只属于自己的本质不同的字串)。

```
struct GSAM{
	int son[N<<1][26];
	int len[N<<1],link[N<<1];
	int siz[N<<1],tot[N<<1],Topu[N<<1],cnt;
	int T[N<<1];
	void init(){
		link[0]=-1;
	}
	int insert(char c,int last,int id){
		int now=last,sonid=c-'a';
		if(son[now][sonid]){
			int q=son[now][sonid];
			if(len[q]==len[now]+1){
				if(T[q])T[q]=-1;
				else T[q]=id;
				return q;
			}
			else{
				int cop=++cnt;
				link[cop]=link[q];
				len[cop]=len[now]+1;
				memcpy(son[cop],son[q],sizeof(son[q]));
				while(now!=-1&&son[now][sonid]==q){
					son[now][sonid]=cop;
					now=link[now];
				}
				link[q]=cop;
				if(T[cop])T[q]=-1;
				else T[cop]=id;
				return cop;
			}
		}else{
			int p=++cnt;
			if(T[p])T[p]=-1;
			else T[p]=id;
			len[p]=len[now]+1;
			while(now!=-1&&son[now][sonid]==0){
				son[now][sonid]=p;
				now=link[now];
			}
			if(now==-1)link[p]=0;
			else{
				int q=son[now][sonid];
				if(len[q]==len[now]+1)link[p]=q;
				else{
					int cop=++cnt;
					link[cop]=link[q];
					len[cop]=len[now]+1;
```

```
                                        memcpy(son[cop],son[q],sizeof(son[q]));
                                        while(now!=-1&&son[now][sonid]==q){
                                                son[now][sonid]=cop;
                                                now=link[now];
                                        }
                                        link[q]=link[p]=cop;

                                }
                        }
                        return p;
                }
        }

        void sort1(){
                for(int i=1;i<=cnt;i++)tot[len[i]]++;
                for(int i=1;i<=cnt;i++)tot[i]+=tot[i-1];
                for(int i=1;i<=cnt;i++)Topu[tot[len[i]]--]=i;
        }
        int ans[N];
        void solv(int n){
                sort1();
                for(int i=cnt;i;i--){
                        int now=Topu[i];
                        //cout<<T[now]<<" "<<i<<endl;
                        if(T[link[now]]&&T[link[now]]!=-1){
                                if(T[now]!=T[link[now]]){
                                        T[link[now]]=-1;
                                }
                        }else if(T[link[now]]!=-1){
                                T[link[now]]=T[now];
                        }
                        if(T[now]!=-1){
                                ans[T[now]]+=len[now]-len[link[now]];
                        }
                }
                for(int i=1;i<=n;i++){
                        printf("%d\n",ans[i]);
                }
        }


}sam;
int n;
char c[N];
int main(){
        n=read();
        sam.init();
        for(int k=1;k<=n;k++){
                scanf("%s",c+1);
```

```
                int len=strlen(c+1);
                int last=0;
                for(int i=1;i<=len;i++){
                        last=sam.insert(c[i],last,k);
                }
        }
        sam.solv(n);
        return 0;
}
```

```
输入
3
amy
tommy
bessie

输出
3
11
19
```

广义后缀自动机（GSAM）

给定 n 个由小写字母组成的字符串 $s_1,s_2,s_3...$，求本质不同的子串个数。（不包含空串）

```
// struct Trie{
//      int son[N][26];
//      int fa[N];
//      int cnt;
//      void insert(char c[],int slen){
//              int now=0;
//              for(int i=1;i<=slen;i++){
//                      int sonid=c[i]-'a';
//                      if(son[now][sonid]==0)son[now][sonid]=++cnt,fa[cnt]=now;
//                      now=son[now][sonid];
//              }
//      }
// }tr;

struct GSAM{
        int son[N<<1][26];
        int len[N<<1],link[N<<1];
        int cnt;
        void init(){
                link[0]=-1;
        }
        int insert(char c,int last){
                int sonid=c-'a';
                if(son[last][sonid]){
                        int now=last,q=son[now][sonid];
                        if(len[q]==len[now]+1)return q;
                        else{
                                int cop=++cnt;
                                link[cop]=link[q];
                                len[cop]=len[now]+1;
                                memcpy(son[cop],son[q],sizeof(son[cop]));
                                while(now!=-1&&son[now][sonid]==q){
                                        son[now][sonid]=cop;
                                        now=link[now];
                                }
                                link[q]=cop;
                                return cop;
                        }
                }else{
                        int now=last,p=++cnt;
                        len[p]=len[now]+1;
                        while(now!=-1&&son[now][sonid]==0){
                                son[now][sonid]=p;
                                now=link[now];
                        }
                        if(now==-1)link[p]=0;
                        else{
                                int q=son[now][sonid];
```

```
                        if(len[q]==len[now]+1)link[p]=q;
                        else{
                                int cop=++cnt;
                                link[cop]=link[q];
                                len[cop]=len[now]+1;
                                memcpy(son[cop],son[q],sizeof(son[cop]));
                                while(now!=-1&&son[now][sonid]==q){
                                        son[now][sonid]=cop;
                                        now=link[now];
                                }
                                link[q]=link[p]=cop;
                        }
                }
                return p;
        }
    }
    //bfs建GSAM，需要Trie
    // int pos[N];
    // void build(){
    //      queue<pair<int,int> >q;
    //      for(int i=0;i<26;i++){
    //              if(tr.son[0][i]){
    //                      q.push({i,tr.son[0][i]});
    //              }
    //      }
    //      pos[0]=0;
    //      while(!q.empty()){
    //              int now=q.front().second,sonid=q.front().first;
    //              q.pop();
    //              pos[now]=insert(sonid,pos[tr.fa[now]]);
    //              for(int i=0;i<26;i++){
    //                      if(tr.son[now][i]){
    //                              q.push({i,tr.son[now][i]});
    //                      }
    //              }
    //      }
    // }
    ll solv(){
        ll ans=0;
        for(int i=1;i<=cnt;i++){
                ans+=len[i]-len[link[i]];
        }
        return ans;
    }
}gsam;
int n;
char c[N];
int main(){
    n=read();
```

```
        gsam.init();
        for(int i=1;i<=n;i++){
                int last=0;
                scanf("%s",c+1);
                int len=strlen(c+1);
                for(int j=1;j<=len;j++){
                        last=gsam.insert(c[j],last);
                }
        }
        ll ans=gsam.solv();
        printf("%lld\n%lld\n",ans,gsam.cnt+1);
        return 0;
}
```

最小表示法

```
int ans[maxn];
void min_rp(int c[],int len){
        for(int i=1;i<=len;i++){
                c[len+i]=c[i];
        }
        int l=1,r=2,k=0;
        for(int i=1;l<=len&&k<=len&&r<=len;i++){
                int opt=c[l+k]-c[r+k];
                if(opt==0){
                        k++;continue;
                }
                else if(opt>0){
                        l+=k+1;
                }else{
                        r+=k+1;
                }
                if(l==r)l++;
                k=0;
        }
        l=min(l,r);
        r=l+len;
        for(int i=l;i<=r;i++){
                ans[i-l+1]=c[i];
        }
}
int n;
int c[maxn];
int main(){
        n=read();
        for(int i=1;i<=n;i++){
                c[i]=read();
        }
        min_rp(c,n);
        for(int i=1;i<=n;i++){
                printf("%d ",ans[i]);
        }
        return 0;
}
```